Maintaining Connectivity in a Scalable and Robust Distributed Environment

Jelasity, M.; Preuss, M; van Steen, M.R.; Paechter, B.

published in
Workshop Global and Peer-to-Peer Computing on Large Scale Distributed Systems
2002

document version
Publisher's PDF, also known as Version of record

Maintaining Connectivity in a Scalable and Robust Distributed Environment*

Márk Jelasity
Department of Artificial Intelligence,
Free University of Amsterdam
De Boelelaan 1081a, 1081 HV Amsterdam,
The Netherlands, jelasity@cs.vu.nl
and RGAI, University of Szeged, Hungary

Mike Preuß
Chair of Systems Analysis, Department of
Computer Science, University of Dortmund
Joseph-von-Fraunhoferstr. 20, 44227 Dortmund,
Germany,
mike.preuss@uni-dortmund.de

Maarten van Steen
Free University of Amsterdam,
steen@cs.vu.nl

Ben Paechter
Napier University, 10 Colinton Road, Edinburgh,
Scotland, EH10 5DT

Abstract

This paper describes a novel peer-to-peer (P2P) environment for running distributed Java applications on the Internet. The possible application areas include simple load balancing, parallel evolutionary computation, agent-based simulation and artificial life. Our environment is based on cutting-edge P2P technology. We introduce and analyze the concept of long term memory which provides protection against partitioning of the network. We demonstrate the potentials of our approach by analyzing a simple distributed application. We present theoretical and empirical evidence that our approach is scalable, effective and robust.

1. Introduction

This paper describes a novel framework for running distributed experiments on the Internet. It is being developed as part of the DREAM project [9]. In a nutshell, the aim of the DREAM project is to develop a complete environment for developing and running distributed evolutionary computation experiments on the Internet in a robust and scalable fashion. The present work focuses on the network engine, i.e. the overlay network on which these experiments will eventually be run.

Although our project focuses on evolutionary computation the environment supports any application that

- is massively parallelizable
- uses asynchronous communication
- has little communication between its subprocesses
- has large resource requirements
- and is robust (the success of the application does not depend on the success of any given subprocess).

This list might seem quite restrictive but in fact it includes many interesting fields. Good examples are running independent tasks with load balancing, island models in evolutionary computation (EC), heuristic optimization, modeling swarm intelligence and other systems with emergent behaviour, etc.

In essence we relax strict requirements concerning reliability of computations and synchronization and control of subprocesses. This allows us to apply scalable P2P technology based on epidemic protocols that can be used on unreliable WANs. This approach has the advantage of being able to access a potentially huge amount of idle resources.

To our knowledge, using a P2P network that deploys epidemic protocols for distributing computational tasks in a fully decentralized manner is new. Existing P2P systems are mainly used for data-oriented applications management like maintaining discussion groups or to distribute information (e.g. [4, 6, 1]). Current systems that use WANs for solving computational problems generally deploy a server/worker paradigm that requires central components, which may lead to scalability or availability problems. (e.g. [3, 10, 11]). The Java platform offers a natural way to distribute computational tasks by allowing runtime linking of executable code to an application. It provides rich security features and at last but not least complete *platform independence*. This made Java an obvious choice for us.

To summarize: our environment can be thought of as a virtual machine or *distributed resource machine* (DRM) made up of computers anywhere on the Internet. The actual set of machines can (and generally will) constantly change and can grow immensely without any special intervention. Apart from security considerations, anyone having access to the Internet can connect to the DRM and can either run his/her own experiments or simply donate the spare capacity of his or her machine.

The outline of the paper is as follows: Section 2. discusses

*This work is funded as part of the European Commission Information Society Technologies Programme (Future and Emerging Technologies). The authors have sole responsibility for this work, it does not represent the opinion of the European Community, and the European Community is not responsible for any use that may be made of the data appearing herein.

name	this is the unique key
address	the IP address and port of the node
date	timestamp of the entry
agents[ ]	names of agents living at the node
map	optional information in a hash map

Table 1. Structure of an entry in the database of a node.

the DRM from an algorithmic and theoretical point of view. We will illustrate the scalability and robustness of the underlying epidemic protocol.

Section 3. gives simulation results for large networks. We show a shortcoming of the algorithm suggested in [5] and suggest and analyze a solution.

Section 4. describes an application developed for our environment. This application performs executes a set of independent tasks with load balancing over the nodes of the network. While this is only a simple application and does not at all make use of all the possibilities, it is suitable for illustrating the features of the DRM. Section 5. describes the results of our experiments on a real DRM under different circumstances. Section 6. concludes the paper.

2. The distributed resource machine

The DRM is a P2P overlay network on the Internet forming an *autonomous agent environment*. Computations are implemented as multi-agent applications. The exact way an application is implemented in the multi-agent framework is not a priory restricted, although we intend to suggest templates and examples in the future (one of which is discussed in Section 4.) to facilitate development.

2.1. Self-organizing structure

The DRM is a network of DRM nodes. In the DRM every node is completely equivalent. There are no nodes that possess special information or have special functions. Nodes must be able to know enough about the rest of the network in order to be able to remain connected to it and to provide information about it to the agents. Spreading information over and about the network is based on epidemic protocols [2].

Every node maintains an *incomplete* database about the rest of the network. This database contains entries on some other nodes (see Table 1). We call these nodes *the neighbours* of the node. The database is refreshed using a push-pull anti-entropy algorithm. Every node s chooses a living address from its database regularly once within a time interval. An address is *living* if there is a working node s' at that address Then any differences between s and s' are resolved so that after the communication s and s' will both have the union of the two original databases (choosing the fresher item if both contain items with a given key). Besides this, s will receive a fresh item on s' (with a new timestamp of course) and s' will also receive an item on s with the actual timestamp. As mentioned before, the size of the database is limited. This limitation is implemented by keeping only the freshest items that fit in (according to the timestamp in the entries). Note that we assume here that the local time at the different nodes does not differ significantly.

Fortunately, the theoretical and practical results discussed below show that limiting the size of the database does not affect the power of the epidemic algorithm. Essentially the same approach was adopted by [5].

To connect a new node to the DRM one needs only one living address. The database of the new node is initialized with the entry containing the living address only, and the rest is taken care of by the epidemic algorithm described above. Removal of a node does not need any administration at all. Note that a node might even change its IP address and/or port while running, so computers with dynamic IP addresses are also automatically supported without any special modification of the algorithm.

2.2. Theoretical properties

The theory of epidemic algorithms is well known [2]. To apply it to our limited-size databases we have to assume that a given node has an equal probability of being in the database of any other node. In Section 3. we will examine a special case when this assumption does not hold.

Let n be the number of nodes in the network, k the size of the database in each node and let a node initiate exactly one information exchange session in every t seconds.

We know that information spreads very fast over the network *if* the network is connected. But what is the probability that the network is connected?

Let $\mathcal{G}(n,k)$ denote a random directed graph of n nodes in which the outdegree of each node is exactly k and these k arcs go to random nodes. Let $\pi(k,n)$ denote the probability that there is a directed path from a given node to any other node in $\mathcal{G}(n,k)$. The following theorem holds [7]:

Theorem 1 *Consider the sequence of random graphs $\mathcal{G}(n,k_n)$ with $k_n = \log n + c + o(1)$, where c is a constant. We have*

$$\lim_{n\to\infty} \pi(k_n,n) = e^{-e^{-c}}$$

It is notable that $1-\pi(k_n,n) < 10^{-10}$ if $n > 23$. The theorem tells us that for a large network of size n if the size of the database is $k = \log n + c$ where e.g. $c > 23$ we have a connected network with an extremely large probability. For example for $k = 100$ we can have $n \approx 10^{33}$. Empirical analysis shows that the constants predicted by the theorem provide the expected performance [7, 5].

3. Recovery after partitioning

In Section 2.2. it was assumed that a given node has an equal probability of being in the database of any other node. In practice this assumption is often unrealistic. For instance if for some reason a subset of the nodes in the DRM (e.g. the ones within a university intranet) is separated from the

rest of the DRM due to the failure of the underlying Internet connection, then this equal distribution assumption cannot be expected to hold. We show that the DRM (and thus the architecture in [5, 7]) is very sensitive to this problem and we will suggest a cheap and simple solution in the form of a *stochastic long term memory*.

3.1. The partitioning problem

We illustrate the problem through a simple example which we will use later for the simulation experiments as well. Let n be the number of nodes in a DRM. Let us assume that initially the equal distribution assumption is true. At some point a cluster of $n/2$ nodes loses physical connection with the other cluster of $n/2$ nodes while connection is preserved within the clusters. Let us denote these clusters with C_1 and C_2 respectively. This results in a situation when nodes exchange information only with nodes from their own cluster.

Due to lack of space we do not detail this part of our experiments but simulations of up to $n = 10000$ and database size 100 show that within a couple of time steps the connectivity of the network is lost, i.e. the clusters completely forget each other. This also means that after restoring the physical connection between the clusters the DRM is not able to recover its integrity; we end up with two independent DRMs. In real networks this would happen within at most a couple of minutes.

Note that entries are never removed from the databases explicitly based on e.g. availability tests. Items "die out" only when their timestamps are too old to be included into the limited-sized databases. This is a negative side effect of the quick adaptivity of the network which is in fact a major advantage in other situations.

3.2. Stochastic long term memory

Our solution to the partitioning problem is the stochastic long term memory. We add an additional set of addresses (long term memory) to every node beside the database. When the node communicates with a peer (according to the epidemic algorithm) the address of the peer is stored in this set with a given probability p_{ltm}. If the size of the set exceeds a fixed limit, a random element is removed.

The epidemic algorithm picks a random element from the long term memory instead of the database with the same probability p_{ltm}. The idea is that this way old addresses are tried time to time which helps to make the connectivity of the DRM robust to physical connection failures. Note that—unlike approaches based on the underlying physical network topology like [8]—this approch is topology independent.

Let us give some theoretial properties of this solution. Let out(C_1) be the number of long term memory entries in the whole C_1 cluster that point to nodes from cluster C_2. Let c be the size of the long term memory. Let out$(C_1) = m$ at time 0. Let us further assume that the physical connection between C_1 and C_2 is lost at time 0 as well. Then after the t-th cycle of the epidemic algorithm out(C_1) follows a binomial distribution with parameters $B((\frac{c-1}{c})^{tp_{ltm}}, m)$. For example for $p_{ltm} = 0.1$, $m = 100$, $c = 100$ and $t = 1000$ the expected value is still 36.6.

Another interesting question is how much time elapses until the expected value of out(C_1) becomes 1. After some elementary transformations we get the following equation:

$$tp_{ltm} = \frac{\log \frac{1}{m}}{\log \frac{c-1}{c}} \approx c \log m$$

This tells us that the size of the memory is much more important for preserving information than the original amount of information. We will see later that even if out(C_1) is only one, i.e. if only one of the nodes has only one address in its long term memory from cluster C_2 this is often sufficient to restore full connectivity.

Note that the size of memory can be much larger than the size of the database because the memory is never exchanged between nodes (it never travels through the network) and it contains only addresses, no additional information (unlike the database).

We can thus calculate the amount of available information as a function of time during the time interval when the physical connection is missing. But what happens when the physical connection is restored between C_1 and C_2? Tables 2 and 3 give simulation results that answer this question for network sizes 1000 and 10000 respectively. The tables show statistics from 10 runs for each parameter setting with $p_{ltm} = 0.1$. p_{con} is the probability of restoring the connectivity between the two clusters, and t is the average time necessary for this.

The most interesting phenomenon that we can observe is that a very small amount of information is sufficient to recover the network. As little as 1 item is sufficient in almost half of the occasions. Note that for a network size of 10000 and $c = 10$ the long term memories of the nodes in C_1 hold 50000 items altogether. When only 3 of these points to the other cluster we experienced successful recovery in 10 out of the 10 cases.

3.3. A last note

The concept of long term memory can easily be extended by applying more sophisticated data structures and machine learning algorithms. Nodes can build a representation of the DRM while communicating with the many different nodes which can increase the chances of the survival of the DRM, even under very poor conditions of the underlying physical network.

4. The test application

The application itself has two layers. The lower layer is an abstract load balancing framework on top of the DRM. The higher layer is the application consisting of a set of tasks to be executed. The only interesting feature of the task set we used for testing in the present experiment is that every task needs exactly the same amount of resources (CPU time and memory) if run on a single fixed machine but the tasks are sensitive to the resources actually being available.

m =		1	2	3	4	5	6	9	15	27
c = 10	p_{con}	40%	60%	80%	90%	100%	100%	100%	100%	100%
	t	70.25	89.17	27.88	30.89	13.90	27.60	15.30	7.70	5.00
c = 20	p_{con}	40%	60%	100%	90%	90%	100%	100%	100%	100%
	t	106.75	119.67	108.50	51.67	57.78	40.00	20.20	20.10	12.10
c = 50	p_{con}	20%	90%	100%	100%	100%	100%	100%	100%	100%
	t	188.50	200.56	277.60	165.50	88.80	153.20	60.70	73.40	21.80
c = 90	p_{con}	50%	70%	90%	80%	100%	100%	90%	100%	100%
	t	229.40	410.14	209.89	269.38	254.10	186.80	95.22	40.40	39.80

Table 2. Results for a network size of 1000.

m =		1	2	3	4	5	6	9	15	27
c = 10	p_{con}	50%	90%	100%	80%	100%	100%	100%	100%	100%
	t	60.20	65.89	29.40	64.50	43.90	24.70	12.40	7.00	5.80
c = 20	p_{con}	70%	70%	90%	70%	90%	100%	100%	100%	100%
	t	119.43	34.29	65.78	93.14	33.56	60.70	62.80	10.70	12.10
c = 50	p_{con}	50%	80%	80%	80%	90%	100%	100%	100%	100%
	t	126.40	197.13	211.38	69.50	119.89	88.70	68.20	59.80	17.50

Table 3. Results for a network size of 10000.

4.1. The load balancing algorithm

In this paper we chose to consider the simplest possible application on the DRM, a load balancing framework. This framework does not make use of the messaging features of the DRM (at least not on the application level) i.e. the tasks do not communicate with each other. This application suffices for illustrating the reliability, scalability and robustness of our DRM system.

We assume that our application is composed of T tasks that have to be run independently of each other as fast as possible. The tasks have to be distributed efficiently over the available resources in a way that tolerates the unreliability and high communication costs of WANs, and the dynamic nature of the DRM, in the sense that machines can come and go at any time.

Load balancing is based on epidemic algorithms just like the DRM itself. The application starts by initiating an *island* which is implemented an autonomous agent. This island can be started on any node within the DRM. The goal of this island is to complete T tasks. The island achieves this goal by starting to work on a task and at the same time listening to the communications of its host node. When the node where the island can be found exchanges information with another node (according to the epidemic protocol of the DRM) the island checks if the peer node already has an island (recall that the database entry of a node contains information about the agents running there). If not it sends half of its remaining tasks to the peer node in the form of a new island which then runs the same distribution mechanism on the peer node in order to complete its tasks. In this way the tasks "infect" the network. Note that it means that there is at most one island on every node.

When an island arrives at its host node it sends a confirmation message back. This is the only communication that is taking place. It ensures that the processing of the set of tasks sent to another node was at least started. Confirmation of finishing tasks does not make sense since the sender island might not exist anymore at that time. This might result in loosing tasks but this is not a serious disadvantage under our assumptions described in Section 1..

The performance of the nodes does not have to be taken into account when sending out tasks since if the machine is too slow, it will send most of its tasks on nodes that finish earlier. Also, no mechanism is needed to indicate that nodes have become available because they will be infected very quickly anyway by simply communicating with peers. These valuable properties result form the nature of our epidemic protocol underlying the DRM discussed earlier.

4.2. Some theoretical properties

We do not need to develop a separate theory to explain the behaviour of the architecture because results describing the epidemic algorithms apply. In the following we briefly summarize these analogies.

Let T be the number of tasks and n the number of nodes. We have to differentiate between three cases. If $T \ll n$ then task spreading follows the behaviour of the starting phase of information spreading in an epidemic network. If $T \gg n$ then the behaviour of the end-phase of the pull version of anti-entropy is relevant. In this case the expected number of tasks an island has is much more than one. When in such a network an empty node connects to a random peer (according to the epidemic algorithm) it is very likely that this peer will have some tasks to send. Thus it is very unlikely that any node remains empty for a long time.

Finally $T \approx n$ is the most problematic case, as there will

be a moment when a considerable number of islands work only on one task. These islands cover a considerable proportion of the DRM. As a consequence, islands running several tasks will only slowly discover idle nodes. However in the first phase the spreading of tasks across nodes is fast; it slows down only in the end phase. But even in this case if the completion time of an average task is much longer than the time between two information exchanges between the nodes then this disadvantage becomes almost invisible.

5. Empirical results

We implemented two different scenarios on a cluster of workstations to substantiate our claims. For both of them, a fixed number of tasks (here: 1500) was computed.

Optimistic scenario: The experiment is running on an undisturbed cluster, no node is added, deleted or restarted.

Cluster addition scenario: After the experiment has been running for a time, an additional cluster of nodes is added to and deleted from the DRM several times. The added nodes are always empty, i.e. they have no tasks, they do not remember their previous state in the DRM.

We are interested in the performance behaviour of the DRM under these different conditions, and would like to see a speedup factor that does not vary much over the two scenarios. Note that we do not want to show robustness in the sense of getting all of the tasks done, this is hindered by the layout of the load balancing system and would in any case be very difficult to achieve on a large-scale distributed P2P system. From our experiments, we cannot conclude much about scalability of the performance behaviour because the number of machines available for testing has been quite limited.

It must be stated that even for the optimistic scenario it is very difficult, if not impossible, to repeat an experiment in exactly the same way. However, we consider the average results over many experiments relatively stable.

5.1. Optimistic scenario

In this case nodes were run on workstations spread all over Europe. We had machines from Paris, Edinburgh, Amsterdam and Dortmund. The number of nodes is stable for this scenario, the experiment runs undisturbed by external influences. This can easily be confirmed visually from Fig. 1. It also shows that the number of working nodes varies between 9 and 11 until most of the tasks are done. At around 3500 seconds, the task distribution seems to get more difficult, so that re-balancing the system begins to take longer and more nodes remain without work. At this point, 83% of the tasks are done. But as long as the number of tasks available exceeds the number of nodes by far, the DRM can recover from this situation. Near the end, the number of tasks left to compute approaches the number of nodes and the suboptimal behaviour described in Section 4.2. ($T \approx n$) appears as predicted. The slowest of the nodes that still have one task to compute determines the end of the experiment. In this case, all of the 1500 tasks have been executed. Note that the number of active islands differs slightly from the working nodes. That is because islands performing task setup and distribution are considered active, but their node is not considered working during this administration time.

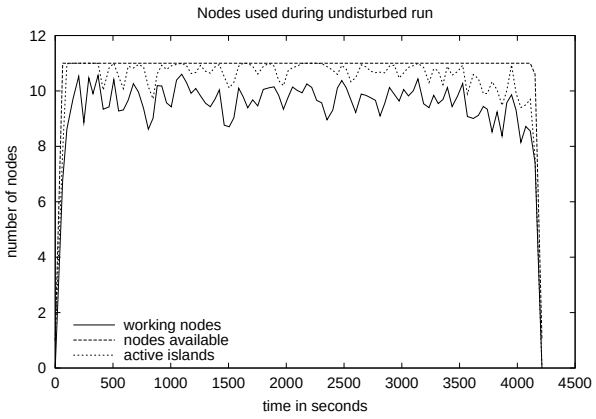

Figure 1.

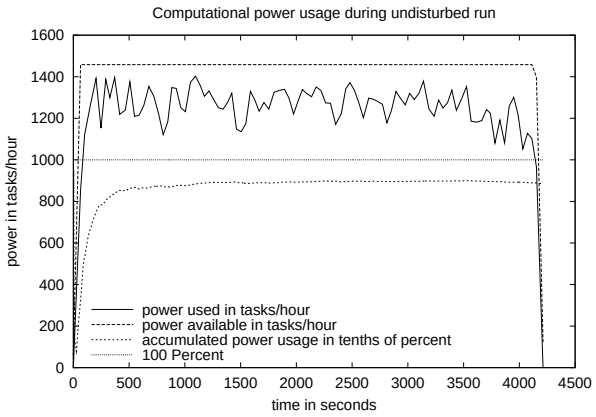

Figure 2.

Figure 2 shows a very similar structure. It displays the used resources relative to the available capacity in terms of tasks per hour. These numbers are determined by using the tasks themselves as a benchmark and computing the approximate maximum speed of a node via the average time needed to finish a task. The accumulated power usage shown as a separate line proves that the available total capacity of all nodes in the experiment is used to more than 86%.

5.2. Cluster addition scenario

Here we used the same cluster as in the optimistic scenario. The additional cluster was located in Dortmund entirely within a single LAN but it contained workstations with highly diverse performances.

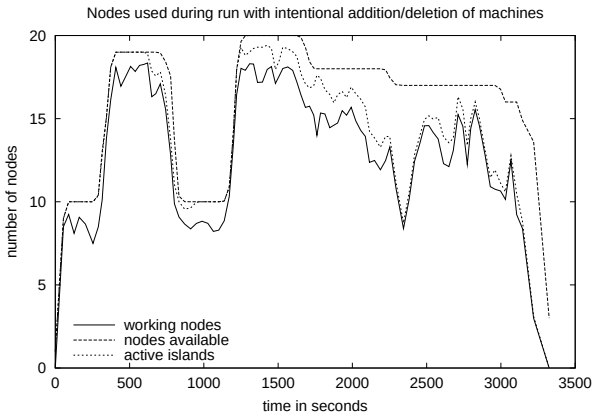

Figure 3.

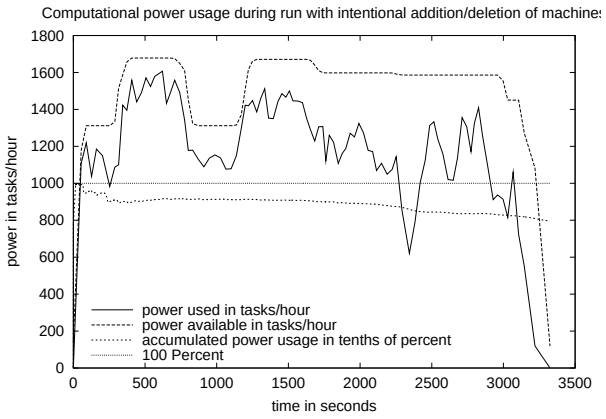

Figure 4.

It can be visually perceived from Figure 3 that 9 additional nodes have been added to the DRM after around 300 seconds. They are quickly found and exploited by placing new islands on them. After 750 seconds have elapsed, the nodes are removed again. This is repeated with 10 nodes later on, this time removing them step by step and not at once. Despite the expectation that this scenario depicts a very extreme course, the DRM copes quite well with the situation. Available resources are utilized rapidly and even the deletion of half of the nodes does not hinder the experiment from continuing.

For this experiment, the difference between the two types of charts (Figures 3 and 4) is clearer. The reason is that the capacity of the added nodes is lower than the capacity of the starting nodes. This is indicated by the smaller steps visible in Fig. 4. Both charts suggest however that most of the available resources are used. At the end, the accumulated power usage is 80%. It is however important to note that not all tasks are actually completed in this scenario. As the islands own their tasks after confirmation (they are not memorized anywhere else in the DRM), the tasks of a prematurely shut down island are lost. Thus, the number of tasks completed in this experiment is only 1104 of the 1500.

6. Conclusions

In this paper we discussed a distributed P2P environment for running special distributed applications from domains like evolutionary computation, social modeling, artificial life, etc.

The concept of long term memory was introduced. Simulation results on large networks were presented together with theoretical considerations which show us that the proposed architecture is stable even if the underlying network is partitioned for a long time.

Empirical results on a real network were also presented. Probably the simplest possible application (load balancing of a given number of independent tasks) was chosen to illustrate the potentials of the system. The application reacted rapidly to changes in the system resulting in good load balancing. High utilization of available resources was also observed.

References

[1] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In H. Federrath, editor, *Designing Privacy Enhancing Technologies*, volume 2009 of *LNCS*, pages 46–66, 2000.

[2] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database management. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC'87)*, pages 1–12, Vancouver, Aug. 1987. ACM.

[3] distributed.net. http://distributed.net/.

[4] P. Druschel and A. Rowstron. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, Banff, Canada, 2001. ACM.

[5] P. T. Eugster, R. Guerraoui, S. B. Handurukande, A.-M. Kermarrec, and P. Kouznetsov. Lightweight probablistic broadcast. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN 2001)*, Göteborg, Sweden, 2001.

[6] Gnutella. http://gnutella.wego.com/.

[7] A.-M. Kermarrec, L. Massoulié, and A. J. Ganesh. Probablistic reliable dissemination in large-scale systems. Submitted for publication, available as http://research.microsoft.com/ camdis/PUBLIS/kermarrec.ps.

[8] M.-J. Lin and K. Marzullo. Directional gossip: Gossip in a wide area network. In J. Hlavicka, E. Maehle, and A. Pataricza, editors, *Dependable Computing – EDCC-3*, volume 1667 of *LNCS*, pages 364–379, 1999.

[9] B. Paechter, T. Bäck, M. Schoenauer, M. Sebag, A. E. Eiben, J. J. Merelo, and T. C. Fogarty. A distributed resoucre evolutionary algorithm machine (DREAM). In *Proceedings of the Congress on Evolutionary Computation 2000 (CEC2000)*, pages 951–958. IEEE, IEEE Press, 2000.

[10] SETI@home. http://setiathome.ssl.berkeley.edu/.

[11] United Devices^{tm}. http://ud.com/.